IFAL
Campus Maceió

Reestruturação de um Programas usando Subprogramas

Objetivo: O objetivo desta atividade é permitir que os alunos pratiquem a reestruturação de um programa escrito de forma procedimental, utilizando subprogramas (funções, procedimentos e métodos). Ou seja, vocês devem identificar as partes do código que podem ser encapsuladas em subprogramas, criar esses subprogramas de forma apropriada e chamar no programa principal.

Isso ajudará vocês a entenderem a **modularização do código** e a escreverem programas mais organizados e fáceis de manter.

Observações:

- Identifiquem áreas do código que realizam tarefas específicas, como entrada de dados, realização de cálculos e exibição de resultados, e transformá-las em funções separadas.
- Tenham atenção a interface dos subprogramas. Escolham nomes descritivos para suas funções, de forma que o propósito de cada função seja claro. Escolham parâmetros necessários e retornos corretos. Lembrem-se que outras pessoas desenvolvedoras podem utilizar esse seu subprograma.
- Cada método deve ter **uma responsabilidade** única e clara.

1. Algoritmo 1: Calculadora (Linguagem C)

Descrição do Programa Original: O programa original é um simulador de uma calculadora simples que realiza operações básicas (adição, subtração, multiplicação e divisão) com dois números inseridos pelo usuário. O programa solicita ao usuário que escolha uma operação, em seguida, insere os dois números e exibe o resultado da operação escolhida.

```c
#include <stdio.h>
int main() {
	char operador;
	float num1, num2, resultado;
	printf("Digite a operação (+, -, *, /): ");
	scanf("%c", &operador);
	printf("Digite o primeiro número: ");
	scanf("%f", &num1);
	printf("Digite o segundo número: ");
	scanf("%f", &num2);
	switch (operador) {
	case '+':
		resultado = num1 + num2;
		printf("Resultado: %.2f\n", resultado);
		break;
	case '-':
		resultado = num1 - num2;
		printf("Resultado: %.2f\n", resultado);
		break;
	case '*':
		resultado = num1 * num2;
		printf("Resultado: %.2f\n", resultado);
		break;
	case '/':
		if (num2 != 0) {
		resultado = num1 / num2;
		printf("Resultado: %.2f\n", resultado);
		} else {
		printf("Erro! Divisão por zero.\n");
		}
		break;
	default:
		printf("Operador inválido.\n");
	}
	return 0;}
```

2. Algoritmo 2: Caixa Eletrônico (Linguagem Java)

O programa original é um simulador de caixa eletrônico simples. Ele permite ao usuário fazer saques, depósitos e verificar o saldo da conta. O programa mantém o saldo em uma variável global e solicita ao usuário que insira sua operação desejada (saque, depósito ou consulta de saldo) e o valor, se aplicável.

```
import java.util.Scanner;
public class CaixaEletronico {
    static double saldo = 1000.00;
    public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
       int opcao;
       double valor;
       System.out.println("Bem-vindo ao Caixa Eletrônico");
       System.out.println("Seu saldo atual é: R$" + saldo);
       System.out.println("Escolha uma opção:");
       System.out.println("1 - Saque");
       System.out.println("2 - Depósito");
       System.out.println("3 - Consultar Saldo");
       opcao = scanner.nextInt();
       if (opcao == 1) {
              System.out.println("Digite o valor do saque:");
              valor = scanner.nextDouble();
              if (valor > saldo) {
              System.out.println("Saldo insuficiente.");
              } else {
              saldo -= valor;
              System.out.println("Saque de R$" + valor + " realizado.");
              }
       } else if (opcao == 2) {
              System.out.println("Digite o valor do depósito:");
              valor = scanner.nextDouble();
              saldo += valor;
              System.out.println("Depósito de R$" + valor + " realizado.");
       } else if (opcao == 3) {
              System.out.println("Seu saldo atual é: R$" + saldo);
       } else {
              System.out.println("Opção inválida.");
       }

       scanner.close();
    }
}
```

3. Algoritmo 3: Agenda (Linguagem Haskell)

O programa original em Haskell é um sistema de gerenciamento de agenda. Ele permite ao usuário armazenar e visualizar eventos. O programa tem as seguintes funcionalidades:

1. Adicionar um evento à agenda.
2. Remover um evento da agenda.
3. Visualizar todos os eventos na agenda.
4. Sair do programa.

```haskell
import Data.List

-- Definição de tipos
type Evento = String
type Agenda = [Evento]

-- Função principal
main :: IO ()
main = do
    putStrLn "Bem-vindo ao Gerenciador de Agenda!"
    loop []

-- Função de loop para interação com o usuário
loop :: Agenda -> IO ()
loop agenda = do
    putStrLn "\nSelecione uma opção:"
    putStrLn "1. Adicionar evento"
    putStrLn "2. Remover evento"
    putStrLn "3. Visualizar agenda"
    putStrLn "4. Sair"
    opcao <- getLine
    case opcao of
        "1" -> do
            putStrLn "Digite o evento a ser adicionado:"
            evento <- getLine
            let novaAgenda = agenda ++ [evento]
            loop novaAgenda
        "2" -> do
            putStrLn "Digite o índice do evento a ser removido:"
            indice <- getLine
            let novaAgenda = delete (agenda !! (read indice)) agenda
            loop novaAgenda
```

```
        "3" -> do
            putStrLn "Eventos na Agenda:"
            mapM_ putStrLn agenda
            loop agenda
        "4" -> putStrLn "Saindo do programa..."
        _   -> do
            putStrLn "Opção inválida. Tente novamente."
            loop agenda
```